

# Clipping of Arbitrary Polygons with Degeneracies

Erich L Foster\* and James Overfelt†

November 15, 2012

## Abstract

Polygon clipping is a frequent operation in Arbitrary Lagrangian-Eulerian methods, Computer Graphics, GIS, and CAD. In fact, clipping algorithms are said to be one of the most important operations in computer graphics.[7, 1] Thus, efficient and general polygon clipping algorithms are of great importance. Greiner et al.[3] developed a time efficient algorithm which could clip arbitrary polygons, including concave and self intersecting polygons. However, the Greiner-Hormann algorithm does not properly handle degenerate cases, without the undesirable need for perturbing vertices.[5] We present an extension to the Greiner-Hormann polygon clipping algorithm which properly deals with degenerate cases. We combine the method proposed by Kim et al.[5] and the method mentioned by Liu et al.[6] to remove or properly label degenerate cases. Additionally, the algorithm presented avoids the need for calculating midpoints, doesn't require additional entry/exit flags, and avoids changing the vertex data structure used in the original Greiner-Hormann algorithm, which was required by the extension presented by Kim et al.[5]

## 1 Introduction

Polygon clipping is an indispensable tool in computer graphics, computer aided design (CAD) [3, 5], geographic information systems (GIS)[6], and even the modelling of fluid dynamics (e.g. Arbitrary Lagrangian-Eulerian methods). In the case of graphics and fluid dynamics, polygon clipping may need to be done thousands of times, and in GIS the polygons that are to be clipped are generally non-convex [7] and therefore efficient and general algorithms for polygon clipping are important.

Greiner et al.[3] developed a simple efficient algorithm that could deal with non-convex and self-intersecting polygons. The advantage of the Greiner-Hormann algorithm lies in its simplicity, generality, and as compared to the Vatti algorithm, its speed[2]. Liu et al. was able to improve the memory efficiency of the original Greiner-Hormann algorithm by maintaining a single doubly linked

---

\*Department of Mathematics, Virginia Tech

†CSRI, Sandia National Labs/NM

list of intersections, instead of places intersections in both polygons.[6] While the memory efficiency giving by this method is certainly important for many applications we are not concerned with this improvement and therefore will only discuss the originally Greiner-Hormann algorithm. While the Greiner-Hormann algorithm has advantages over similar algorithms it does suffers from degenerate cases. Greiner et al. suggested a perturbation of these degeneracies.[3, 6] However, this leads to undesirable effects such as different solutions for different perturbation directions.[5]

In the sections that follow we present an extension to the Greiner-Hormann algorithm for clipping arbitrary 2D polygons with degeneracies. The Greiner-Hormann algorithm has been lauded as having the simplest data structure and outperforming similar polygon clipping algorithms such as Vatti.[5, 2] However, there is one serious flaw in the Greiner-Hormann algorithm: degenerate case. Greiner et al. suggested perturbing polygon vertices to deal with degenerate cases, but as demonstrated by Kim et al. this is undesirable due to the fact that the resultant clipping is highly dependent upon the perturbation direction, thereby making the algorithm indeterminate.[5] Kim et al. presented their own extension to the Greiner-Hormann algorithm that deals with these degenerate cases without the need for perturbing polygon vertices. However, the method the Kim extension requires calculating midpoints for certain degeneracies. In what follows, we present an algorithm which combines the method developed by Kim et al.[5] and a method of removing intersection mention by Liu et al.[6] This method, as compared to the Kim et al. method, avoids additional entry/exit flags, changing the original vertex data structure used in the Greiner-Hormann algorithm, and the calculation of midpoints. The calculation of midpoints can be complicated when dealing with spherical polygons.

Additionally, the extension we present does not require any additions to the original traversal algorithm used by the Greiner-Hormann. However, the time efficiency is hampered by requiring that all intersections be evaluated for labelling, whereas the original Greiner-Hormann algorithm only required the first intersection be evaluated for labelling and then all subsequent labels were set by toggling the previous label.

## 2 Greiner-Hormann

For completeness we present the original algorithm describe by Greiner et al. using a simple example. The Greiner-Hormann algorithm consists of three phases: (1) Compute points of intersection between the subject polygon,  $S$ , and the constraint polygon,  $C$ ; (2) Set traversal flags at the intersections; (3) Traverse the polygons adding vertices or intersections, to the clipped polygon, as they are encountered.

In the Greiner-Hormann algorithm polygons are presented as circular doubly linked lists, where each node represents a vertex or an intersection. Each vertex-intersection is represented by a *vertex data structure*, see Figure 1. Subject and constraint polygons are linked through intersections by a neighbor pointer. This

```

vertex = {
    real x,y;           /*coordinates*/
    vertex *next, *prev; /*link pointers*/
    bool isect;          /*intersection*/
    bool en;             /*entry or exit*/
    vertex *neighbor;    /*pointer to intersection
                           in adjacent polygon*/
    vertex *nextPoly     /*pointer to head of next
                           polygon*/
}

```

Figure 1: Vertex Data Structure

neighbor pointer points to the corresponding intersection in the neighbor polygon. To differentiate between intersections and vertices, a boolean flag, *isect*, is set to *True* for intersections and *False* for vertices. Additionally, intersections use a boolean flag, *en*, to indicate if an intersection is an entry, *True*, or an exit, *False*. The same boolean flag is used for vertices to indicate if the vertex is inside, *True*, or outside, *False*, of the neighbor polygon. It is this *en* flag and the neighbor pointer that are the crux of the Greiner-Hormann algorithm, and tells one which direction to travel on the polygon and allows for switching between polygons.

We will not discuss the first phase in the Greiner-Hormann algorithm, intersections, since this has been covered extensively by others and may differ depending upon the setting, e.g. spherical polygons.[2, 5] However, the second phase, which deals with labelling intersections, is of great importance and is where our extension of Greiner-Hormann will differ. Greiner et al. traverses each polygon once, marking each vertex as an entry, *en*, or an exit, *ex*. Initially, one traverses a given polygon until the first intersection is encountered, at this point the intersection is determined to either be an entry or an exit to the neighbor polygon and label as such. One then continues to traverse the given polygon until another intersection is encountered labelling it the opposite of the previous intersection, i.e. *ex* if the previous intersection was an *en* and *en* if the previous intersection was an *ex*. Once all intersections have been labelled we proceed to the third phase.

The third phase of the Greiner-Hormann algorithm deals with the traversal and building of the clipped polygon. To begin the traversal of the clipped polygon one begins at the first vertex of the subject polygon and traverse it until an intersection is encountered; this intersection is subsequently added to the clipped polygon. Traversal direction is then determined by the entry/exit flag of the corresponding intersection; if the intersection is an entry one proceeds forward along the subject polygon, but if the intersection is an exit one proceeds in the backward direction along the subject polygon. While traversing a polygon all vertices are added to the clipped polygon. Once a new intersec-

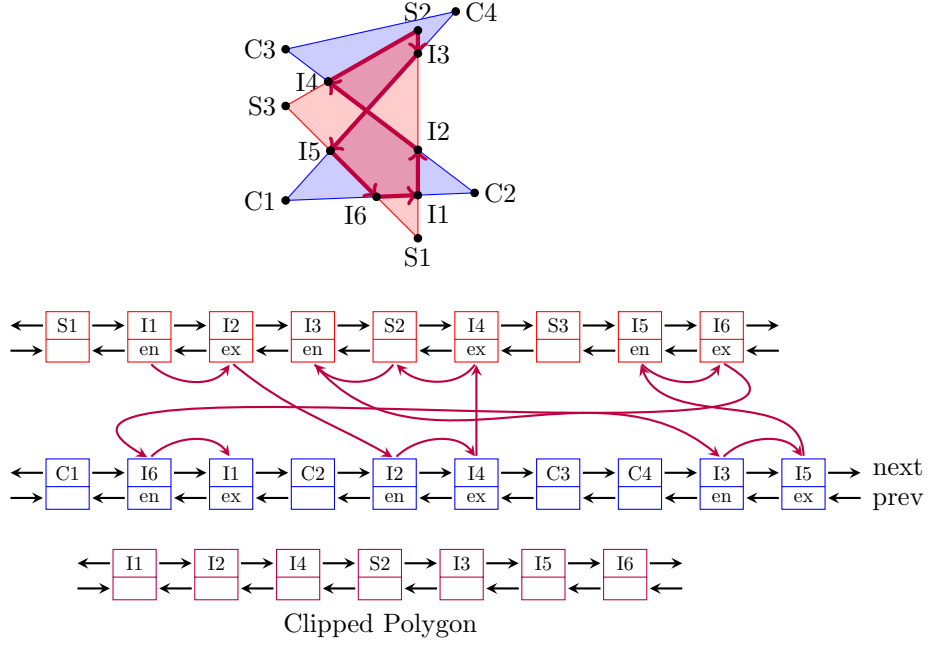


Figure 2: Demonstration of the Greiner-Hormann Algorithm

tion is encountered it is added to the clipped polygon and one switches, using the neighbor pointer, to the adjacent polygon. Again, direction of traversal on the adjacent polygon is determined by the entry/exit flag. This procedure is repeated until one reaches the first vertex of the clipped polygon. The overall procedure is continued, creating new clipped polygons as needed, until all intersections are consumed. A demonstration of the Greiner-Hormann algorithm including the traversal process can be seen in Figure 2.

### 3 Degeneracies

In Section 2 we assumed there were no degenerate cases, i.e. cases where a vertex lies on the edge of the neighbor polygon. Greiner et al. suggested perturbing these vertices to resolve degeneracies.[3, 6] The problem with this method of perturbation is clear and was even demonstrate in Figure 12 of [3]; the method of perturbation can result in different solutions depending upon the direction of perturbation, i.e. perturbing to the outside or inside of the neighbor polygon (see Figure 3).[5] The differing results means this method is indeterminate and not appropriate for various applications, such as climate modelling.

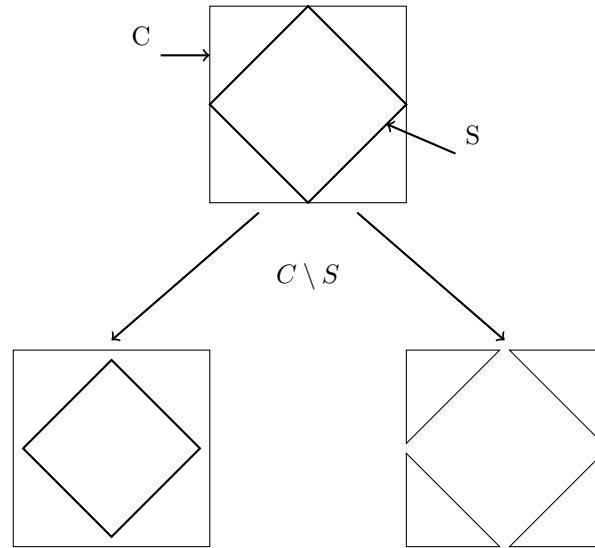


Figure 3: Example of perturbation method and varying results depending on direction of perturbation.

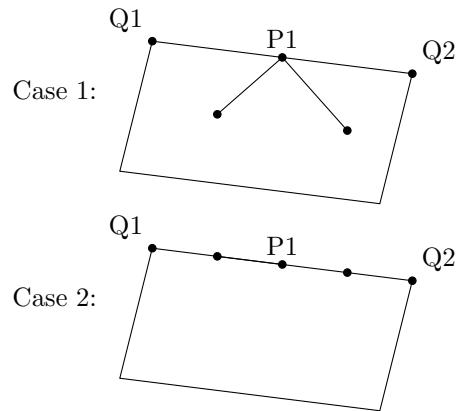


Figure 4: Degenerate cases

## 4 Extension of Greiner-Hormann Algorithm

In this section we discuss the extension of the Greiner-Hormann algorithm by a combination of setting traversal flags and removing unneeded/unwanted intersections. Greiner et al. took advantage of the entry/exit property, whereby intersections flip between entry, *en*, or exit, *ex*, as one moves from intersection to intersection. However, this assumption doesn't always hold true. Unfortunately, this is one of the properties of the Greiner-Hormann algorithm which makes it efficient, and therefore the dismissal of this assumption will make the algorithm's efficiency suffer. At the same time, it is this assumption that results in a lack of robustness in the Greiner-Hormann algorithm. So, while our extension is more robust it will not be as efficient as the original Greiner-Hormann algorithm.

The main difference in our method as compared to that used in the original Greiner-Hormann method is that we do not make the assumption that one can determine the entry/exit flag of an intersection from the previous intersection. In fact, we use the location (inside/outside/on) of the previous and next vertex in the polygon data type to determine the entry/exit property of an intersection. The inside/outside/on property can be determined using a ray casting algorithm such as the one presented by Hormann et al. [4] Additionally, our method differs from the method used by Kim et al. in that we do not introduce any new entry/exit flags (i.e. *ex/en*, *en/ex*), the vertex data structure from the Greiner-Hormann algorithm remains unchanged, we do not require the calculation of midpoints to determine flags, and the starting location of the traversal (i.e. the initial intersection of the clipped polygon) does not matter. Thus, our extension of the Greiner-Hormann algorithm only requires a change in the vertex labelling phase of the original algorithm, and doesn't change the other phases. Thus the implementation of our extension only requires recoding the labelling phase.

As in the standard Greiner-Hormann algorithm, polygons are clipped in three stages; (1) compute intersections between the subject and constraint polygons, (2) set traversal flags of intersections and remove appropriate intersections if needed, (3) traverse the polygons switching between polygons at intersections and adding vertices/intersections, to the clipped polygon, as they are encountered.

### 4.1 Intersections

The process of adding intersections is unchanged from the Greiner-Hormann algorithm, and, again, we will not discuss the method of finding these intersections since the method may differ from situation to situation and has been discussed extensively by others.[2, 5] However, recall from the original Greiner-Hormann algorithm that once an intersection is discovered it is added to the subject and constraint polygon. Also, recall that each intersection is connected to its corresponding intersection in the neighbor polygon through the neighbor pointer. It is this neighbor pointer which allows one to switch back and forth between polygons.

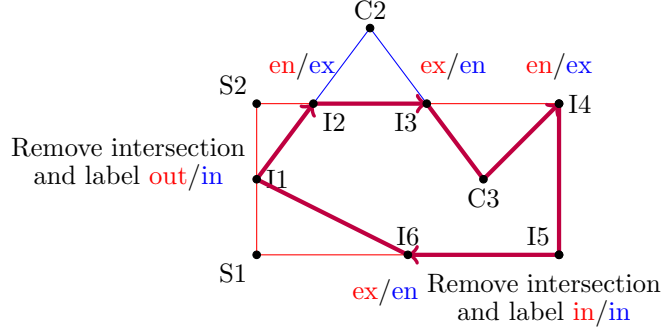


Figure 5: Example of labelling/Removal process.

## 4.2 Labelling/Removal of Intersections

The process of setting entry/exit flags or removing intersection flags consists of: first, cycling through the vertices of both the subject and constraint polygons and labelling them as inside or outside; second, one cycles through the intersections of the subject polygon labelling or removing intersections appropriately. One only needs to traverse the subject polygon, since the same intersections exists in both polygons and the intersection in the constraint polygon can be accessed through the neighbor pointer. Once a label is set for an intersection it is then required to check the (current,neighbor) entry/exit pair before moving onto the next intersection, this way a degenerate intersection does not affect the labelling of the next intersection. An example of this process can be seen in Figure 5. It should also be noted that all intersections are assumed to be initially set to *en*. This is necessary to prevent one intersection removal from preventing a second intersection removal, which would occur in Figure 3 if intersections were initially set to *ex*.

The previous and next pointers associated with an intersection can be one of three cases: (1) *on*, the point is on the edge of the neighbor polygon, i.e. is another intersection; (2) *in*, the point lies within the neighbor polygon; (3) *out*, the point lies outside the neighbor polygon. Therefore, there are a total of nine cases to deal with when setting traversal flags or removing intersection flags. The cases (in, out) and (out, in) are the original cases dealt with by the Greiner-Hormann algorithm and their entry/exit flag remains unchanged, i.e. *ex* and *en*, respectively.

However, there are the degenerate cases that must also be dealt with. For intersections, which have (prev,next) pairs (out,out), (in,in), and (on,on) the action required is dependent upon the neighbor's pairing. If the neighbor also is of one of the three aforementioned cases, we will remove the intersection label, i.e. set *isect* to *False*, since these do not add any new information to the traversal. If the neighbor is not of these three cases the current intersection is labelled either *ex* or *en* for (out,out) and (in,in) respectively. For the case of (on,on)

prev / next	on	out	in
on	<sup>1,2</sup>	<i>ex</i>	<i>en</i>
out	<i>en</i>	<i>ex</i> <sup>1,3</sup>	<i>en</i>
in	<i>ex</i>	<i>ex</i>	<i>en</i> <sup>1,4</sup>

Table 1: Action and label for each case.

1. Remove if neighbor is also one of the cases (on,on), (out,out), or (in,in). 2. Label opposite of neighbor if removed otherwise label *in*. 3. Label *out* if removed. 4. Label *in* if removed.

current / neighbor	ex	en
ex	remove and label <i>out</i>	unchanged
en	unchanged	remove and label <i>in</i>

Table 2: Action after label has been set

the intersection is labelled the opposite of its neighbor, e.g. if the neighbor is (in,out) the current intersection would be labelled *en*. In addition to the cases already mentioned, one must be able to label cases where the (prev,next) pair consists of one on. In the case of (on,out) or (in,on) the intersection is labelled *ex*, while cases of (on,in) or (out,on) are labelled *en*.

Once a label is set, one must then check the (current,neighbor) entry/exit flag pair to ensure that one of the (en,en) or (ex,ex) cases does not occur. If an (en,en) pair is encountered the intersection flag is removed and the vertex is set to *in*, on the other hand if an (ex,ex) pair is encountered the intersection flag is also removed, but the vertex is labelled *out*. For the other two cases (en,ex) and (ex,en) the intersection remains unchanged. For a summary of actions and labels refer to Table 1 and Table 2.

### 4.3 Clipping

The algorithm for traversal remains unchanged from the original Greiner-Hormann algorithm and therefore no changes to the original code is needed. Thus, when an intersection is encountered one switches to the neighbor and proceeds forward for an *en* flag and backward for an *ex* flag. The traversal is complete when one reaches the initial clipped polygon vertex. This overall process is continued, creating new clipped polygons as needed, until all intersections are consumed.

It should be noted that for the example given in Figure 3, and cases like it, all intersections would have been removed and one would check to see if either polygon was completely contained within the other. For the subject polygon, *S*, given in this example, all vertices were also intersections, which would have been labelled as *in* when the intersection flag was removed. Thus, looping through vertices of the polygon, *S*, and checking if all vertices are labelled as *in* is sufficient for checking for a nested polygon.

## 5 Conclusions

We effectively dealt with degenerate cases, which was a limiting factor for the use of the Greiner-Hormann algorithm. The intersection labelling technique employed is less efficient than the original method used by Greiner et al., however degenerate cases are no longer a problem. By combining the methods developed by Kim et al. and Liu et al. we are able to avoid, as compared to the Kim et al. extension, adding additional entry/exit flags, changing the vertex data structure from the original Greiner-Hormann algorithm, and calculating midpoints, which can be costly and complicated for spherical polygons. Of course, the labelling method presented does require checking neighbor intersection labels and possible subsequent removal of an intersection, which may be costly in its own right, especially when polygons have many sides and many intersections.

## References

- [1] Max Agoston. Clipping. In *Computer Graphics and Geometric Modeling*, pages 69–110. Springer London, 2005.
- [2] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*, chapter 2, pages 19–43. Springer Berlin Heidelberg, 3rd edition, 2008.
- [3] Gunther Greiner and Kai Hormann. Efficient clipping of arbitrary polygons. *Association for Computing Machinery-Transactions on Graphics*, 17(2):71–83, 1998.
- [4] Kai Hormann and Alexander Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20:131–144, 2001.
- [5] Dae Hyun Kim and Myong-Jun Kim. An extension of polygon clipping to resolve degenerate cases. *Computer-Aided Design & Applications*, 3(1-4):447–456, 2006.
- [6] Yong Kui Liu, Xiao Qiang Wang, She Zhe Bao, Matej Gombosi, and Boru Zalik. An algorithm for polygon clipping, and for determining polygon intersections and unions. *Computers & Geosciences*, 33:589–598, 2007.
- [7] Antonio Schettino. Polygon intersections in spherical topology: Applications to plate tectonics. *Computers & Geosciences*, 25:61–69, 1999.